BY JOSEPH WEISBECKER

Build the COSMAC "ELF" Microcomputer

PART 3: How to expand memory, plus more programs

IN TWO previous articles (POPULAR ELECTRONICS, August 1976 and September 1976), we discussed the construction of the low-cost Elf microcomputer, gave some programming examples, and described some low-cost optional input/output circuits. Here we will examine some software operating systems and discuss adding 1024 bytes of memory for as little as $20.

Operating Systems. An operating system is a program that makes it easier to program and use your computer. For example, if you want to change M(43) in the basic Elf memory, you would have to start at M(00) and step through memory to location 43 before you could change it. Program 1 is a simple operating system for the Elf microcomputer. It lets you directly examine or modify any memory location. It also allows you to start program execution at any memory location. We call Program 1 ETOPS-256 (Elf Toggle OPerating System for 256-byte memory). After loading ETOPS in RAM, it can be used to help you load and run other programs.

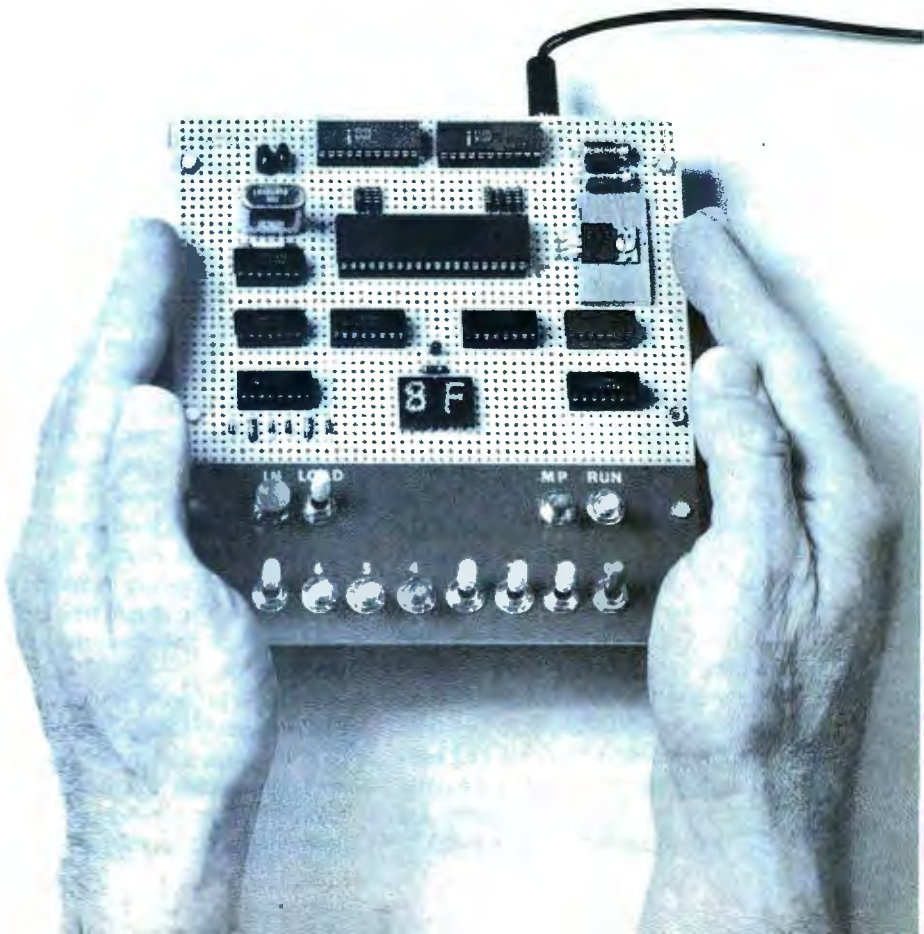
To examine a memory location using ETOPS, set 01 into the toggles. Flip the RUN switch up and 01 will be displayed. Now set the address of the memory byte you want to examine into the toggles and push the INPUT switch. The next time you push the INPUT switch, you'll see the selected memory byte displayed. Keep pushing the INPUT switch to see the sequence of bytes stored in memory.

To modify any memory location, set 02 into the toggles and turn the RUN switch up. 02 will appear. Set the address of the memory byte you want to modify (via the toggles).

Push the INPUT switch and the Q light comes on. Now set the toggles to the value of the byte you want to place in the selected memory location and push the INPUT switch to store it in RAM. You can store a sequence of new bytes by setting each byte into the toggles and pushing the INPUT switch. The Q light warns that you are modifying memory.

If you have the toggles set to 00 when you flip the RUN switch up, you can then set the toggles to the beginning address of a program you want to execute. Just push the INPUT switch to start executing your program at the selected address. Your program will begin execution with R3 as the program counter.

If you've added the battery RAM option to your system, ETOPS will be ready to use as soon as you turn on power. Since ETOPS uses only 32



PROGRAM 1--ETOPS-256

```
0000   F8  20  A1    R1.0 = work
  03   E1          X=1
  04   6C  64  21    D = toggles
  07   3F  07        Wait for IN on
  09   37  09        Wait for IN off
  0B   32  1D        M(1D) if D=00
  0D   F6  33  11    M(11) if D=01
  10   7B            Q=1
  11   6C  A1        R1.0 = toggles
  13   3F  13        Wait for IN on
  15   37  15        Wait for IN off
  17   39  1A        M(1A) if Q=0
  19   6C            M1 = toggles
  1A   64            Show M1, R1+1
  1B   30  13        Repeat M(13)
  1D   6C  A3        R3.0 = toggles
  1F   D3            P=3
  20   00            Work area
  21   User programs from
         M(21) to M(FF)
```

PROGRAM 2--EHOPS-256

```
0000   F8  FF  A2    R2.0 = work
  03   F8  23  A5    R5.0 = BSUB
  06   F8  33  A6    R6.0 = HSUB
  09   F8  0D  A3    R3.0 = M(0D)
  0C   D3            P=3
  0D   D5  A1        BSUB, R1.0=D
  0F   6C            D, M2 = toggles
  10   3A  14        M(14) if D≠00
  12   81  A3        R3.0 = R1.0
  14   F6  3B  1C    M(1C) if D=02
  17   D5  E1        BSUB, X=1
  19   64            Show M1, R1+1
  1A   30  17        Repeat M(17)
  1C   D5  E1        BSUB, X=1
  1E   51  64        M1=D, show M1, R1+1
  20   30  1C        Repeat M(1C)
  22   D3            P=3 (return)
BSUB
  23   D6            HSUB
  24   FE  FE        D left x 2
  26   FE  FE        D left x 2
  28   A0  D6        R1=D, HSUB
  2A   80  F1  52    M2=R1 or M2
  2D   64  22        Show M2
  2F   30  22        Go to M(22)
  31   F0  D5        D=M2, P=5
HSUB
  33   E2  FC  01    X=2, D+1
  36   FA  0F  52    M2=D and 0F
  39   62  22        Select key M2
  3B   3D  33        M(33) if key off
  3D   7B  F8  09    Q=1, D=09
  40   B4            R4.1=09
  41   24  94        R4-1
  43   3A  41        M(41) if R4.1≠00
  45   7A            Q=0
  46   35  46        Wait for key off
  48   30  31        Go to M(31)
```

PROGRAM 3

```
0050   F8  FF  A1      R1.0 = work
  53   F8  00  51      M1=00
  56   E1  64  21      Show M1
  59   F0  FC  01  51   M1+1
  5D   F8  10  B2      R2.1 = delay
  60   22               R2-1
  61   92  3A  60       M(60) if R2.1≠00
  64   30  56           Repeat M(56)
```

bytes, you still have 224 bytes available for your own programs.

Keyboard System. Adding a hex keyboard would make your Elf microcomputer even easier to use. With 16 keys labelled 0 through F, you would have to press only two keys for each byte you want to store in memory. In the second article, we described a circuit for monitoring the states of 16 switches or keys. (See POPULAR ELECTRONICS, Sept. 1976, page 38, Fig. 3). If you add this circuit and a 16-key hex keyboard, you can use Program 2—EHOPS-256 (Elf Hex OPerating System for 256-byte memory). This program requires 74 bytes of RAM so you still have 182 bytes left for your own programs. You can also use the hex keyboard subroutine as part of your programs if desired.

After loading EHOPS in memory, you can use it as follows. To load a byte into any memory location from the hex keyboard, set the toggles to 02 and flip the RUN switch up. The 02 toggles tell EHOPS that you want to store bytes in memory. On the hex keyboard, press the most-significant digit of a memory address followed by the least-significant digit. This address byte will be displayed and tells EHOPS where you want to start loading bytes in memory. You can now load a sequence of bytes into memory via the hex keyboard. Just press the two digits (most significant first) of each byte you want to load and they will be stored sequentially in memory starting at the selected location.

To examine any memory location (without changing its contents), set the toggles to 01 before you flip the RUN switch up. Using the hex keyboard, enter the one-byte starting address of the sequence of memory locations you want to examine. Press any hex key twice to step through memory and display the stored bytes.

To run a program you've loaded using EHOPS, set the toggles to 00 before flipping the RUN switch up. Using the hex keyboard, enter the one-byte starting address of your program. It will begin running with R3 as the program counter.

EHOPS controls the hex keyboard with two subroutines called BSUB and HSUB. BSUB calls HSUB by changing the program counter to R6 with a D6 instruction. HSUB continuously scans all 16 hex keyswitches until one is pressed. It provides a switch debounce delay and waits until the key has been released. It then returns control to BSUB with the value of the pressed key in the least-significant digit of the byte in D and M2.

BSUB is called by changing the program counter to R5 with a D5 instruction. It waits until two hex keys have been pressed before returning control to the calling program with the values of the two keys in the two digits of the byte in D and M2. The most-significant digit represents the first key pressed. Any program you write with R3 as the program counter can call BSUB to obtain a byte from the hex keyboard. If you drive a speaker with the Q lines as described in the September article, you will hear an audible click each time a hex key is pressed.

Program 3 can be loaded and run us-

```
                PROGRAM 4

*C0  F8  F0  AA   RA.0=F0
 C3  F8  08  A8   R8.0=08
 C6  D5  5A       BSUB, MA=D
 C8  1A  28       A+1, R8-1
 CA  88  3A  C6   M(C6) if R8.0≠00
*CD  F8  F0  AA   RA.0=F0
 D0  F8  08  A8   R8.0=08
 D3  EA  F0  A7   R7.0=MA
 D6  64  28       Show MA, A+1, 8-1
 D8  F8  FF  AC   RC.0=FF
 DB  7B  87       Q=1, D=R7.0
 DD  FF  01       D-01
 DF  3A  DD       M(DD) if D≠00
 E1  7A  87       Q=0, D=R7.0
 E3  FF  01       D-01
 E5  3A  E3       M(E3) if D≠00
 E7  2C  8C       RC-1
 E9  3A  DB       M(DB) if RC.0≠00
 EB  88  3A  D3   M(D3) if R8.0≠00
 EE  30  CD       M(CD) if R8.0=00
 F0-F7 = Table of tone values
```

```
                PROGRAM 5

0000  F8  00  B1        R1.1=00
  03  F8  FF  A1        R1.0 = work
  06  F8  00  51        M1=00
  09  E1  64  21        Show M1
  0C  F0  FC  01  51    M1+1
  10  F8  10  B2        R2.1 = delay
  13  22                R2-1
  14  92  3A  13        M(13) if R2.1≠00
  17  30  09            Repeat M(09)
```

ing either ETOPS or EHOPS: This program continuously counts up at a rate determined by the byte at M(5E). Be sure to start execution at M(50).

Program 4 should be loaded and run using EHOPS. You should also have a speaker attached to the Q line. Start this program at M(C0) with EHOPS. You can then enter eight bytes via the hex keyboard. These bytes should have values between 02 and 7F for best results. Each byte represents the frequency of a tone you will hear via the speaker. After you enter the eighth byte you'll hear the eight-tone sequence repeated over and over. You can restart the program at M(CD) to hear a previously entered tone sequence.

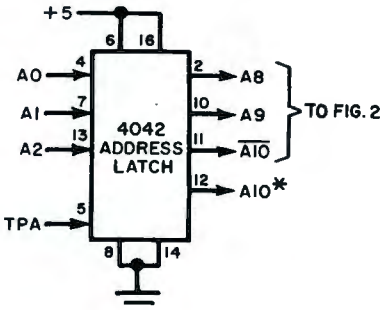


*Fig. 1. Address latch. *Connect pin 19 of original 2101 RAM's to A10 instead of ground.*

An operating system can be designed to incorporate any desired feature. For example, you might want to examine the contents of internal 1802 registers or control the operation of a cassette recorder. As more features are needed,

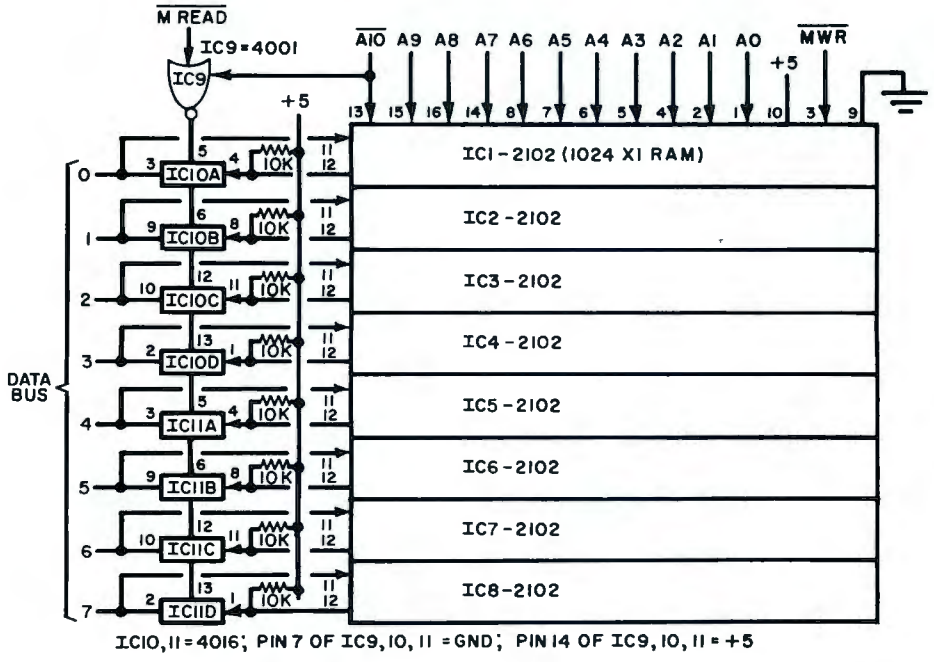


Fig. 2. Eight low-cost, readily available 2102 RAM's (1024 x1) and two transmission gate packages.

you may want to dedicate the entire 256 bytes of memory in the basic system to your operating system and add another section of memory for your other programs. The 256-byte operating-system memory can be battery-powered and protected from modification by the MP switch so that it is always ready for use.

Memory Expansion. You can add 1024 bytes of memory to an Elf microcomputer using inexpensive, readily available 2102-type static RAM's as shown in Figs. 1 and 2. The 10k bus pull-up resistors are required if the high-output level of the RAM chips isn't at least 3 volts. Bits 0 and 1 of the high-order address byte are clocked into the address latch with TPA (Fig. 1). These two latched bits are used with the low-order COSMAC address byte to provide the required 10-bit address for the 2102 RAM's. Bit 2 of the high-order address byte is clocked into the address latch for use in selecting either the original 256-byte RAM or the added 1024-byte section of RAM. Disconnect pin 19 of the original two 2101 RAM chips from ground and connect to pin 12 of the 4042 address latch in Fig. 1.

The original 256-byte memory will now be addressed as 0000-00FF and the new 1024-byte memory will be addressed as 0400-07FF. Since all of the previous programs assumed one-byte addresses, they will *not* run in this expanded memory system. Programs for systems with more than 256 bytes of memory must have both the high-order and low-order bytes of address registers properly set. The previous programs can be easily modified to run in the expanded system by initializing both high- and low-order bytes of any 16-bit register used to address memory. The foregoing counting program could be modified to run at M(0000) in an expanded RAM system as shown in Program 5. In general, it adds only a few bytes to program for an expanded-memory system. By adding bits to the address latch of Fig. 1, you could address up to 64k bytes of RAM. Instead of addressing extra memory, the high-order address bits could be used to select input/output circuits or devices.

Don't forget that adding memory will increase system power requirements. As the system is expanded, make sure your external power supply can handle the increased current requirements. With this in mind, you'll find that the Elf can be tailored to your needs at low cost. ◇

A READER'S ELF PROGRAMS

I recently constructed the COSMAC Elf described in your August (1976) issue and thoroughly enjoyed the construction and testing of this microprocessor system. I build approximately two projects a month that are illustrated in your magazine—plus some from other sources. This particular project turned out to be the most interesting I have ever constructed. Here are three programs that I found useful in illustrating various system functions.

Program I is simply an expansion of your Q-light program with additional decisions that alternately turn the Q light on and off when the input switch is depressed.

Program II displays and increments successive hex characters each time the input button is depressed. To do this, it was necessary to learn how to input to and output from the memory, using pointers in registers, and also to do simple arithmetic through the accumulator (D register).

Program III plays SOS in Morse code. The program should be loaded through the system switch registers if you have a half hour without interruption. With this program, registers are used for pointers to subroutine loops set up for time delay. Three subroutines for 0.5 second, 1 second and 3 seconds are established, addressed by changing the program counter. The main program simply turns the Q light on and off at intervals determined by the subroutines. The memory provided in the basic Elf system (256 bytes) is enough for approximately 19 code elements. Each code element requires only 10 instructions for an on and off interval in the main program. The timing loops require the use of two registers to provide a sufficient time. In my Elf, I used a 1-MHz crystal. Obviously, changing one instruction in the loop subroutines will vary the time as necessary. Changing or adding to the main program can change the code.

Try loading this program with the switch register if you have enough patience.

—Robert Klein

PROGRAM I

```
SWITCH ON AND OFF          3F
                           ØØ
                           37
                           Ø2

IF Q OFF GO TO Ø9          39
                           Ø9

IF Q ON, TURN OFF AND      7A
RETURN TO ØØ               3Ø
                           ØØ

IF Q OFF, TURN ON AND      7B
GO TO ØØ                   3Ø
                           ØØ
```

PROGRAM II

```
STORE DEPENDENT VARIABLE ØØ    E4
IN LOCATION 77 WITH POINTER    F8
IN R4--DESIGNATE R4 AS RX      77
                               A4
                               F8
                               ØØ
                               54

STORE INDEPENDENT VARIABLE Ø1  F8
IN LOCATION 76 WITH POINTER    76
IN R5                          A5
                               F8
                               Ø1 (size of INCR)
                               55

DISPLAY AND DECREMENT RX       64
                               24

LOOK FOR INPUT SWITCH ON AND   3F
OFF                            ØF
                               37
                               1Ø

ADD TWO VARIABLES AND PUT      Ø5
RESULT IN LOCATION 77          F4 (F5 subtract)
(can be changed to subtract    54
to count down)

RETURN TO START OF LOOP        3Ø
                               Ø7
```

PROGRAM III

MAIN PROGRAM

INITIALIZE POINTERS	F8 65 * A3 F8 79 * A4 F8 8D * A5	THIRD DOT	7B D3 F8 65 * A3 7A D3 F8 65 * A3	THIRD DASH	7B D4 F8 79 * A4 7A D3 F8 65 * A3	FIFTH DOT	7B D3 F8 65 * A3 7A D3 F8 65 * A3		
FIRST DOT	7B D3 F8 65 * A3 7A D3 F8 65 * A3	FIRST DASH	7B D4 F8 79 * A4 7A D3 F8 65 * A3	FOURTH DOT	7B D3 F8 65 * A3 7A D3 F8 65 * A3	SIXTH DOT, PAUSE AND RETURN TO START	7B D3 F8 65 * A3 7A D5 F8 8D * A5 3Ø ØØ		
SECOND DOT	7B D3 F8 65 * A3 7A D3 F8 65 * A3	SECOND DASH	7B D4 F8 79 * A4 7A D3 F8 65 * A3						

* If a different number of code elements is used, change the starting address of each sub routine, or move to the end of memory page if flexibility is desired.

PROGRAM III

SUB ROUTINES (Must be loaded in order indicated after main program is loaded.)

	½-Sec Loop	1-Sec Loop	3-Sec Loop
		All instructions same as ½-Sec Loop except where indicated	
PUT 256 IN REG #1	F8 FF A1	.	.
PUT VARIABLE INTO REG #2	F8 08 *	ØF *	3Ø *
DECREMENT AND LOOP THRU R1 UNTIL ZERO. THEN OUTPUT TO DECREMENT REG #2	A2 C4 C4 C4 C4 C4 21 81 3A	.	.
LOOP BACK TO START R #1 CYCLE UNTIL TOTAL TIME IS USED UP	6B ** 22 82 3A 6B **	7F **	93 **
RETURN TO MAIN PROGRAM	DØ	.	.

* Sets Time

** If a different number of code elements is used, change this instruction to starting address of each subroutine wait loop (first C4)